Kumaran Systems Inc.



Cloud based DevOps Lifecycle

Implementation Approach Document

Table of Contents

[1. Introduction 3](#_Toc504582989)

[2. Context 3](#_Toc504582990)

[3. Implementation Approach 3](#_Toc504582991)

[Activity 1: DevOps Lifecycle and Tooling 3](#_Toc504582992)

[Scenario 1:Additional elements required when developing with micro-services and containers. 3](#_Toc504582993)

[Additional elements For developing .NET Microservices: 5](#_Toc504582994)

[Additional elements For developing Java Microservices: 5](#_Toc504582995)

[Activity 2: Pipeline and workflow for modern development 6](#_Toc504582996)

[Scenario 1:Pipeline and workflow 6](#_Toc504582997)

[For .NET 6](#_Toc504582998)

[For Java 7](#_Toc504582999)

[Scenario 2: Implement Tooling for CI/CD 8](#_Toc504583000)

[.Net tooling implementation 8](#_Toc504583001)

[Java tooling implementation 9](#_Toc504583002)

[Scenario 3: Tooling for automation – Infrastructure 10](#_Toc504583003)

[Scenario 4: Tooling for automation – Application 10](#_Toc504583004)

[For .Net 10](#_Toc504583005)

[For Java 11](#_Toc504583006)

[Activity 3: Containerization approach 11](#_Toc504583007)

[Scenario 1: Approach to utilizing containers 11](#_Toc504583008)

[Scenario 2 : Deploy container(s) 11](#_Toc504583009)

[Deployment Architecture: 12](#_Toc504583010)

[Scenario 3 : Scale containers 12](#_Toc504583011)

[Activity 4: Develop micro service(s) 13](#_Toc504583012)

[Scenario 1: Develop and deploy one or more microservices for the application 13](#_Toc504583013)

[Application Flow: 13](#_Toc504583014)

[Activity 5: API Management 17](#_Toc504583015)

[Scenario 1: Design APIs for Microservices development 17](#_Toc504583016)

[Assumptions 17](#_Toc504583017)

# Introduction

Honda needs to understand how to implement a Cloud based DevOps lifecycle that supports global development teams using Java or .Net. This document describes the implementation approach and the recommended tools to implement the DevOps lifecycle.

# Context

The objective of this document is to provide Honda a deeper understandingof the following in a PaaS context:

1. Implementing a cloud based DevOps lifecycle that supports global development teams using Java and/or .Net
2. Integrating Microservices development into the DevOps pipeline and the implementation approach in CSP.
3. Implementing containers into the pipeline and implementation approach in CSP.
4. Implementing API Management (in general and for Microservices specifically).

# Implementation Approach

## Activity 1: DevOps Lifecycle and Tooling

### Scenario 1:Additional elements required when developing with micro-services and containers.

***Scenario Description : Create an approach to developing applications/services using microservices and containers.***

#### Infrastructure Automation

* Infrastructures will be maintained in Cloud
* On Demand scaling of infrastructure is possible
* No Upfront hardware cost

#### Configuration Management

* On demand Server configuration and Software installation is possible.
* Managing Servers/Software are easy using wide range of Configuration tools.

#### Source Code Management

* GIT/SVN are widely used Source Code Repositories
* Repositories will be created and maintained for Projects Source codes and Configuration files management.
* Code duplication is avoided through Versioning
* Source code will be managed in Master branch
* New Sub branches are created below Master branch for CRs and Bug fixes.

#### Deployment Management

* Facilitates Continuous Integration with Gated Check-in
* Code Quality check, Unit and Functional Testing are integrated

#### Release Management

* Automated and Semi Automated approvals for Code Releases
* Provisioning Application deployment to different environments based on requirement

#### Log Management

* Performs unified log processing for all environments.
* Logs can be application or server specific.
* ELK is a very popular tool used for log processing.

#### Performance Management

* Enables Easy Application performance monitoring
* Alert Notification based on Application availability and usage

#### Monitoring Management

* Metrics analytics & visualization suits for infrastructure
* Alert Notification based on Server availability

***Implementation Approach Flow***

### C:\Users\muth1378.KSPLAD\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\734DVL2J\Microservice_Architecture_Final (2).jpg

### Additional elements For developing .NET Microservices:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Category | Tools | License | Cost (Year) | Quantity | Total Cost |
| **Front-end Tools** | | | | | |
| Source Code IDE | Visual Studio 2017 Professional  (On-Premises - BYOL) | Microsoft Licensed | $540 | 99 | $53,460 |
| Visual Studio 2017 Enterprise  (On-Premises - BYOL) | Microsoft Licensed | $2,999 | 1 | $2,999 |
| Source Repository | Git | Open Source | Free |  | Free |
| CICD Tool | VSTS | Microsoft Azure Licensed | $750 | 100 | $75,000 |
| Unit Test Tools | Microsoft Unit Test | VSTS Component |
| Static Code Analyzer Tools | StyleCop | VSTS Component |
| Functional Testing Tools | Selenium | VSTS Component |
| Issue Tracking Tools | VSTS Work Item | VSTS Component |
| Data visualization Tools | Kibana | Open Source | Free |  | Free |
| Server Monitoring Tool | Grafana | Open Source | Free |  | Free |
| Application Monitoring Tools | Application Insights for Enterprise | Microsoft Azure Licensed | $290.40 | 1 | $290.40 |
| Swagger | API Management | Apache 2.0 License | Free |  | Free |
| **Back-end Tools** | | | | | |
| API Gateway | Ocelot | Open Source | Free |  | Free |
| Container Tools | Dockers | Open Source | Free |  | Free |
| Docker Registry | Azure Container Registry | Microsoft Azure Licensed | $243.455 | 1 | $243.455 |
| Container Services | Azure Container Services | Microsoft Azure Licensed | $1374.96 | 1 | $1374.96 |
| Cluster Nodes | Container VMs | Microsoft Azure Licensed | $2077.44 | 2 | $4154.88 |
| On Prem/Cloud DB | SQL Server | Microsoft Azure Licensed | $1800.78 | 1 | $1800.78 |
| Data + Storage | HDD | Microsoft Azure Licensed | $923.52 | 1 | $923.52 |
| Chef | Configuration Tool | Chef Licensed/Node | $137 | 1 | $137 |

\*BYOL - Bring Your Own License. On Premises Visual Studio IDE license can also be used for Cloud Visual Studio IDE Licensing.

### Additional elements For developing Java Microservices:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Category | Tools | | License | | Cost | | Quantity | | Total Cost | | |
| **Front-end Tools** | | | | | | | | | | | |
| Source Code IDE | Eclipse | | Open Source | | Free | | 100 | | | Free | |
| Source Repository | GitLab Repository | | Free for On-Premises | | Free | |  | | | Free | |
| CICD Tool | Jenkins | | Open Source | | Free | |  | | | Free | |
| Unit Test Tools | JUnit | | Open Source | | Free | |  | | | Free | |
| Static Code Analyzer Tools | SonarQube | | Open Source | | Free | |  | | | Free | |
| Functional Testing Tools | Selenium | | Apache Licensed | | Free | |  | | | Free | |
| Issue Tracking Tools | Jira | | Atlassian Licensed | | $66 | | 100 | | | $6,600 | |
| Data visualization Tools | Kibana | | Open Source | | Free | |  | | | Free | |
| Server Monitoring Tool | Grafana | | Open Source | | Free | |  | | | Free | |
| Application Monitoring Tools | Application Insights for Enterprise | | Microsoft Azure Licensed | | $290.40 | | 1 | | | $290.40 | |
| Swagger | | API Management | | Apache 2.0 License | | Free | |  | | | Free | |
| **Back-end Tools** | | | | | | | | | | | | |
| API Gateway | Zuul | | Open Source | | Free | |  | | | Free | |
| Container Tools | Dockers | | Open Source | | Free | |  | | | Free | |
| Docker Registry | Azure Container Registry | | Microsoft Azure Licensed | | $243.00 | | 1 | | | $243.00 | |
| Container Services | Azure Container Services | | Microsoft Azure Licensed | | $1,374.96 | | 1 | | | $1,374.96 | |
| Cluster Nodes | Nodes-Container VMs | | Microsoft Azure Licensed | | $2,077.44 | | 2 | | | $4,154.88 | |
| On Prem/Cloud DB | SQL Server | | Microsoft Licensed | | $1,800.78 | | 1 | | | $1,800.78 | |
| Data + Storage | HDD | | Microsoft Azure Licensed | | $923.52 | | 1 | | | $923.52 | |
| Chef | | Configuration Tool | | Chef Licensed/Node | | $137 | | 1 | | | $137 | |

## Activity 2: Pipeline and workflow for modern development

### Scenario 1:Pipeline and workflow

***Scenario Description: Move a workload from initiation to deployment using the new pipeline. Exercise the entire pipeline. (major release, minor release and bug fix)***

### For .NET

* Visual Studio is used as development IDE for .Net.
* Code changes and commit will be carried out from IDE.
* Sign into <https://www.visualstudio.com/team-services/> with Cloud Account.
* Create New Account for VSTS Team foundation service.
* Create project repository with Team Foundation Version Control.
* Code commit from Visual Studio through team foundation version control.

#### Build Process

* Build Configuration first we need to setup code level build setup like restore, build and test.
* Also Checking for unit test case, Style cope to get passed for further build process.
* Next we need to build docker image for application using custom agent in vsts.
* Push the image to azure container registry and publish artifact for release management.
* If the build process gets failed it will assign to work item for bug fix in vsts.
* Once the bug is fixed and approved then build process start again and go for release management.
* Automated Build Trigger option will be enabled in Continuous Integration and so, when ever code check-in happens, a new build will be triggered.

#### Minor release

* Once the build process gets completed release process will start and execute functional test automatically.
* If the version is minor, no approval process will be carried out and the image will be directly deployed to DEV Environment.
* If functional test case gets passed it goes for QA team and Test URL will be sent to QA.
* Once QA is successful it deployed into PROD Environment.

#### Major release

* If the version is major, approval process is carried out through mail and once approved; the respective image will be pushed to DEV Environment.
* If functional test case gets passed it goes for QA team and Test URL will be sent to QA.
* Once QA is successful it deployed into PROD Environment.

### For Java

* Pull source code from git to eclipse IDE.
* Code changes and commit will be carried out from IDE.
* Tag based commit is used to differentiate the major and minor release.
* Automated Build Trigger option will be enabled in Continuous Integration and so, when ever code check-in happens, a new build will be triggered.
* Build and deployment instructions will be carried out based on the definitions.

#### Minor release

* Code analysis and Junit test will get executed, on successful execution the image will get deployed to DEV environment.
* If one of the tests fails then deployment to DEV environment will not happen and will be notified in mail with test report.
* After deploying to DEV environment, functional test will get executed, on successful execution the image will get deployed to PROD environment (without approval).
* If functional test fails, then deployment to PROD environment will not happen.
* Functional test failure will be notified in mail with test report.

#### Major release

* Code analysis and Junit test will get executed, on successful execution the image will get deployed to DEV environment.
* If one of the tests fails then deployment to DEV environment will not happen and will be notified in mail with test report.
* After deploying to DEV environment, functional test will get executed, on successful execution the image will get deployed to PROD environment, only after the approval.
* If functional test fails, then deployment to PROD environment will not happen.
* Functional test failure will be notified in mail with test report

### Scenario 2: Implement Tooling for CI/CD

***Scenario Description: Perform a branch merge into the modern pipeline and see it merged and compiled.***

### .Net tooling implementation

* Visual Studio for Source Code Editor.
* VSTS for CI/CD Pipelines
* MSBuild for Build
* TFS for Source Control

#### Tag based commit:

Tag based commits are identified through VSTS Build numbers and we can identify whether it’s major or minor release.

Based on revision numbers our custom condition will trigger the release.

#### Branch based commit:

Branch based commit is done in Visual Studio.

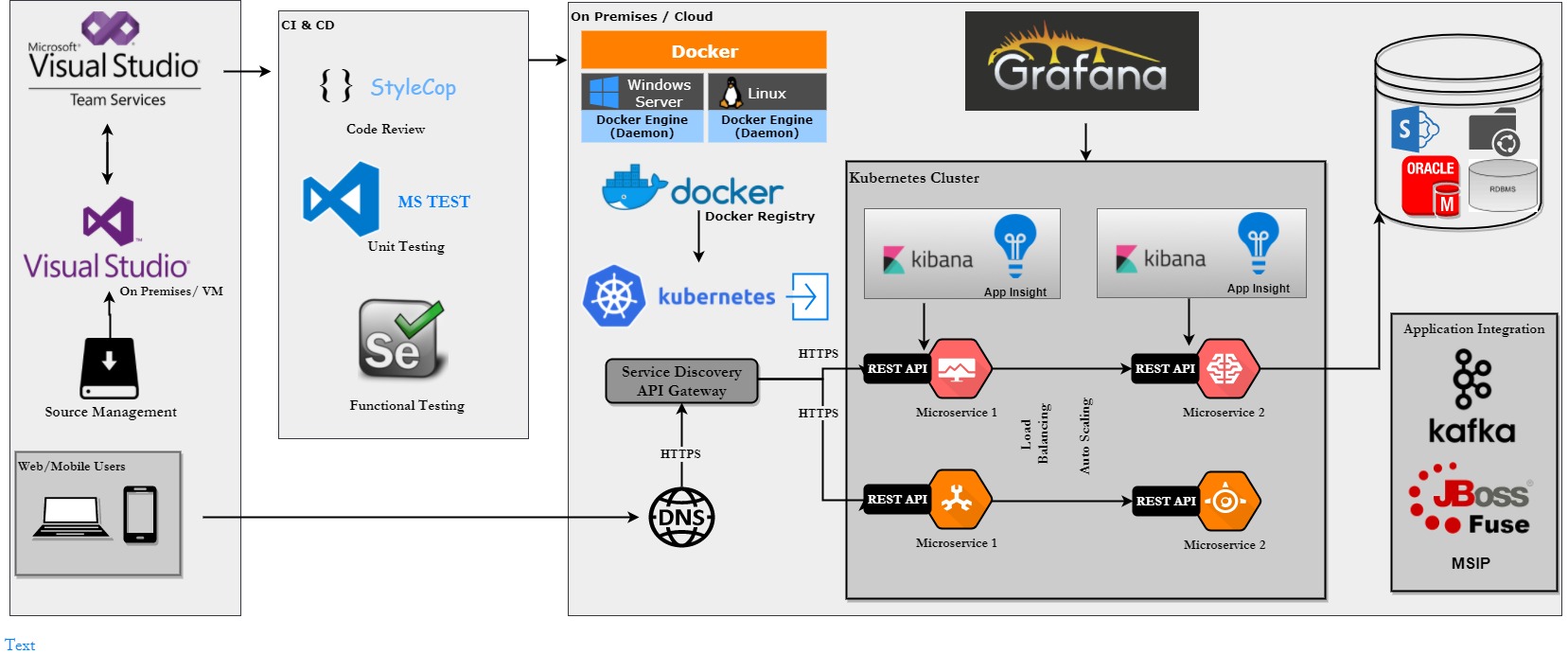
Consider having two branches master and slave.

Once the merge request, arise from the slave branch, the master will merge if required.

Whenever merge operation has been handled, there will be a build trigger in VSTS.

* Selenium for Automation Testing.
* Docker for Containerization.
* Style Cop for Static Code Analysis
* VSTS Mail for Notification & Approvals.

#### .Net Implementation Architecture



### Java tooling implementation

* Eclipse for Source Code Editor.
* Jenkins for CI/CD Pipelines
* Maven for Build
* Git for Source Control
* Checkin/Commits are based on Tag and Branch.

#### Tag based commit:

Tag based commit is done in eclipse to differentiate the major, minor and bug fix release.

We can trigger the build in Jenkins only on the tag based commit event.

#### Branch based commit:

Branch based commit is done in eclipse.

Consider having two branches master and slave.

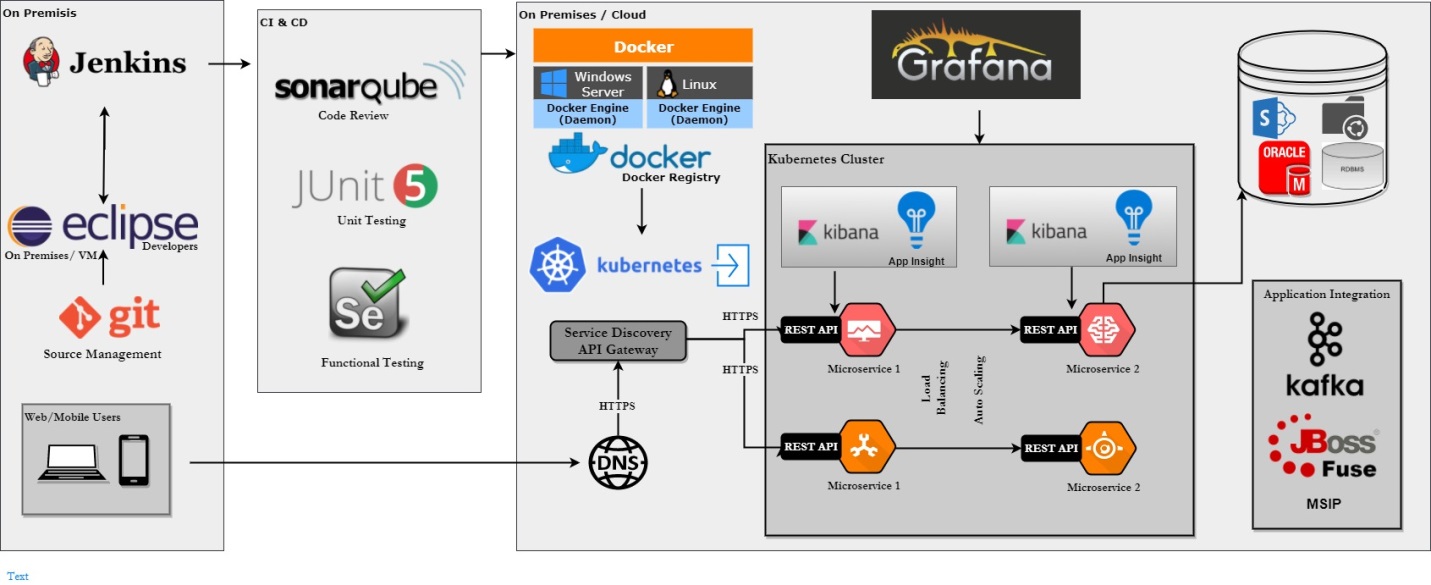
Once the merge request, arise from the slave branch, the master will merge if required.

In Jenkins build trigger has been restricted to merge operation.

Whenever merge operation has been handled, there will be a build trigger in Jenkins.

* Selenium for Automation Testing.
* Docker for Containerization.
* Sonarqube for Static Code Analysis
* Azure mailjet for Notification & Approvals.

#### Java Implementation Architecture



### Scenario 3: Tooling for automation – Infrastructure

***Scenario Description: Move a workload from initiation to deployment using the new pipeline***

* Infrastructure as code Framework will be in place for configuration management.
* Time consuming activity like Manual patching, Configuration updates and service installation for every server can be automated.
* Chef is recommended tool for managing infrastructure by writing code(Automating infrastructure) rather than using manual processes.

### Scenario 4: Tooling for automation – Application

***Scenario Description: Move a workload from initiation to deployment using the new pipeline***

### For .Net

* VSTS will be used as Source Repository.
* Visual Studio is used as IDE, to perform code changes.
* VSTS as CI/CD pipeline tool, which automates the code check-in.
* Style Cop is used as code analyzer.
* MS Test is used for running unit test cases.
* Azure container registry to store docker images.
* Azure container service to deploy docker images to kubernetes cluster.
* Functional Testing will be carried out on post deployment using selenium.
* VSTS Work item is used as automatic issue tracker for any testing and builds issues. Respective user is assigned with a unique issue id.
* An Approval mail will be initiated to deploy the image to Production using VSTS Pre deployment mail approval.
* Grafana will be used to Kubernetes cluster Monitoring
* Kibana will be used to for Application Log Monitoring
* App Insights will be used for Application Performance Monitoring from Azure Portal

### For Java

* Git will be used as Source Repository.
* Eclipse is used as IDE, to perform code changes.
* Jenkins as CI/CD pipeline tool, which automates the code check-in.
* SonarQube is used as code analyzer.
* Junit is used for running unit test cases.
* Azure container registry to store docker images.
* Azure container service to deploy docker images to kubernetes cluster.
* Functional Testing will be carried out on post deployment using selenium.
* Jira is used as automatic issue tracker for any testing and build issues. Respective user is assigned with a unique issue id.
* An Approval mail will be initiated to deploy the image to Production using Azure mailjet.
* Grafana will be used to Kubernetes cluster Monitoring
* Kibana will be used to for Application Log Monitoring
* App Insights will be used for Application Performance Monitoring from Azure Portal

## Activity 3: Containerization approach

### Scenario 1: Approach to utilizing containers

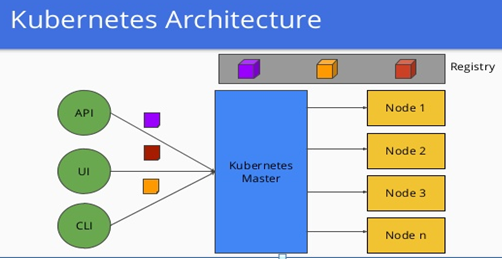
***Scenario Description: Implement docker containers. Define the overall containerization pipeline and workflow and implement it in the cloud.***

* Docker is used as a Container platform
* Azure Container Registry and Azure Container Services are used to Store and deploy the Docker images.
* Source code is pushed to CI/CD pipeline to build Docker images and images are stored in Azure Container Registry
* Release image to Kubernetes Cluster(Azure Container Service) specific to DEV/SIT/UAT/PROD Environments.

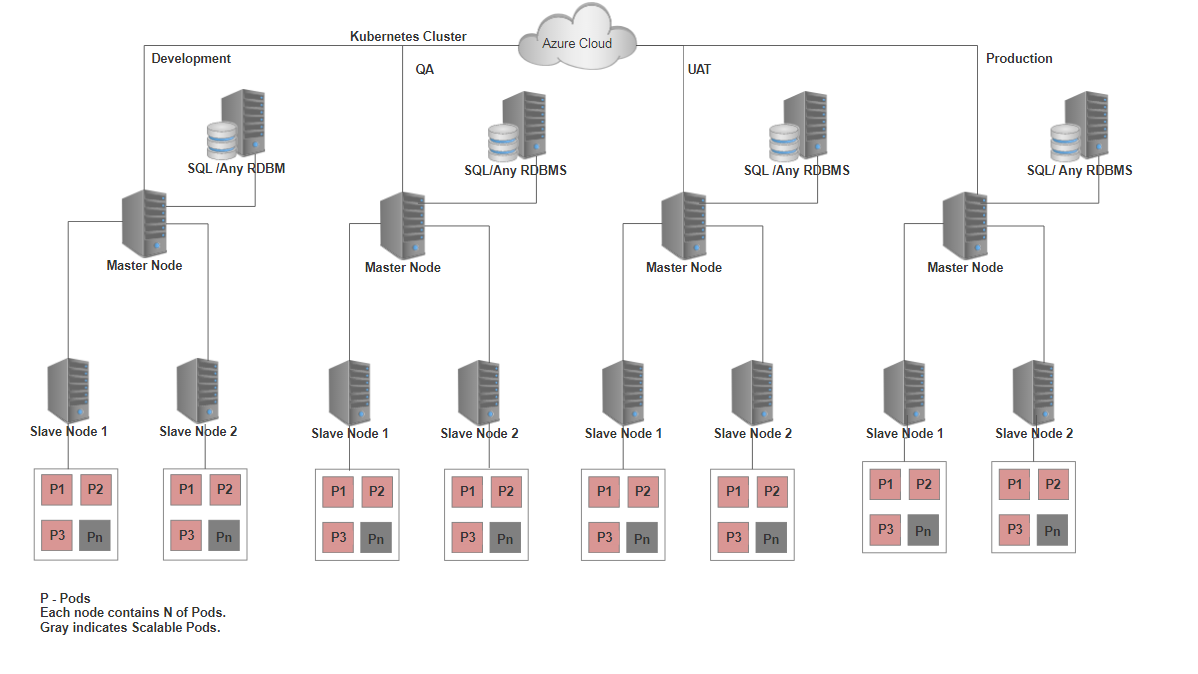
### Scenario 2 : Deploy container(s)

***Scenario Description: Using the pipeline defined in A3S1 deploy container(s)***

* Each Service will have deployment.yaml and service.yaml configuration files.
* When deployment.yaml file is applied in Kubernetes master node, application will be deployed to the Kubernetes cluster. One pod will be created in cluster.
* When service.yaml file is applied in Kubernetes master node, application services will be craeated in the Kubernetes cluster.
* Only Front End Service will be exposed to an External End point. Other Services will be communicated each other using internal ips and ports



### Deployment Architecture:



### Scenario 3 : Scale containers

***Scenario Description: Scale the infrastructure for an app using autoscaling. Given 3 VMs scale the app container from 1 to 2 to 3 through autoscaling.***

* Horizontal Pod Autoscaling(HPA) configuration file is defined with distinct minimum and maximum replicasets.
* Each Application service will have one Horizontal Pod Autoscaling(HPA) configuration file
* Manually enable HPA in Kubernetes Application clusters.
* We recommend choosing an adequate margin between the scale-out and in thresholds. As an example, consider the following better rule combination.
  + Increase instances by 1 count when CPU% >= 80
  + Decrease instances by 1 count when CPU% <= 60
* A Simple Auto scaling calculation:
  + - Assume there are 2 instances to start with.
    - If the average CPU% across instances goes to 80, autoscale scales out adding a third instance.
* Now assume that over time the CPU% falls to 60.
* Autoscale's scale-in rule estimates the final state if it were to scale-in. For example, 60 x 3 (current instance count) = 180 / 2 (final number of instances when scaled down) = 90. So autoscale does not scale-in because it would have to scale-out again immediately. Instead, it skips scaling down.
* The next time autoscale checks, the CPU continues to fall to 50. It estimates again - 50 x 3 instance = 150 / 2 instances = 75, which is below the scale-out threshold of 80, so it scales in successfully to 2 instances.

## Activity 4: Develop micro service(s)

### Scenario 1: Develop and deploy one or more microservices for the application

***Scenario Description: Develop and deloy one or more microservices for the application***

### Application Flow:

C:\Users\sund1871.KSPLAD\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\TK4LDIKU\API_IMAGE_UPDATED.png

#### .NET Microservice Development

#### Tools and frameworks

We have developed monolithic application into micro services based on the following technology.

View – UI are designed from Angular JS

Service – .Net Core REST API project are developed from visual studio.

DAO – Entity framework are used for database like SQL Server / Oracle / DB2 / My SQL

#### Micro services

* User
* OAuth/LDAP Server
* API Gateway
* Mailer

#### User Interface:

* The user will access a Web Application written using Angular JS.
* It will then connect to an OAuth Authorization Server, which will be a central point of where users and authorities can be assigned.
* This server will return a JSON Web Token containing info about the client with its authorities and the grated scope.
* After the user is authenticated and has a token, the Web Application will be able to talk to the API gateway.
* It will take the JWT, verify if it’s coming from the Authorization Server, and then make calls to the micro services and build the response.

#### OAuth/LDAP Server

* The OAuth server uses the User service to get the user’s authentication details. Also, the API gateway uses the OAuth server to get the user’s information.

#### API Gateway

* Ocelot manipulates the HttpRequest object into a state specified by its configuration until it reaches a request builder middleware where it creates HttpRequestMessage object which is used to make a request to a downstream service.
* The middleware that makes the request is the last thing in the Ocelot pipeline. It does not call the next middleware.
* The response from the downstream service is stored in a per request scoped repository and retrived as the requests goes back up the Ocelot pipeline. There is a piece of middleware that maps the HttpResponseMessage onto the HttpResponse object and that is returned to the client. That is basically it with a bunch of other features.

#### Application Flow

* We have defined four micro services to run ARB .Net Core application.
* Web UI service is the only external end points. Also, it will communicate with the other services internally through gateway using nginx path based routing concept.
* Gateway service internally calls Dash Board and User Management services using kubernetes cluster names instead of IPs.
* Dash Board and User Management services have REST Call to Azure SQL Database to access the data and load the data to Web UI service.
* User Management Service performs CRUD operation to manage the Users.

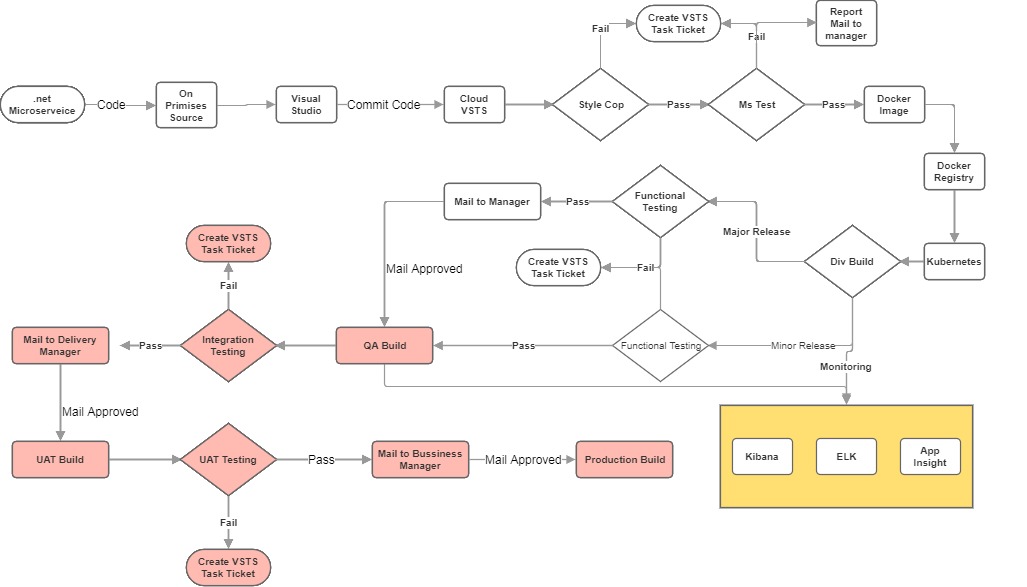
#### Deployment

* Application images are pulled from Azure Container Registry and deployed in Azure Container Services (kubernetes cluster).
* Once the application is deployed, Azure Container service will manage the deployment and services based on the application.
* Auto scaling will be handled from the kubernetes cluster based on the service CPU utilization it will increase or decrease the pods.

#### Deployment Strategies

* Blue/Green deployment reduces downtime and risk by running two identical production environments and at any time, only one of the environments is live, with the live environment serving all production traffic.
* Gated Deployment enables secured code check in which doesn’t get pushed to other environments unless it passes a set of test and also ensures Quality delivery in less turnaround time.

#### Architecture Flow:

****

#### Java Microservice Development

#### Tools and frameworks

We have developed micro services based Application on the following technology.

View – UI are designed from Angular 4

Service – SpringBoot

DAO – JPA framework are used for database like SQL Server / Oracle / DB2 / My SQL

#### Microservices

* User Service
* Mail Service
* Notification Service
* API Gateway
* OAuth/LDAP Server

#### IMAGE

* The user will access a Web Application written using Angular. It will then connect to an OAuth/LDAP Authorization Server, which will be a central point of where users and authorities can be assigned. This server will return a JSON Web Token containing info about the client with its authorities and the grated scope. After the user is authenticated and has a token, the Web Application will be able to talk to the API gateway. It will take the JWT, verify if it’s coming from the Authorization Server, and then make calls to the micro services and build the response.

#### Authentication

* Spring Security and OAuth2 are obvious choices when talking about secure distributed system.

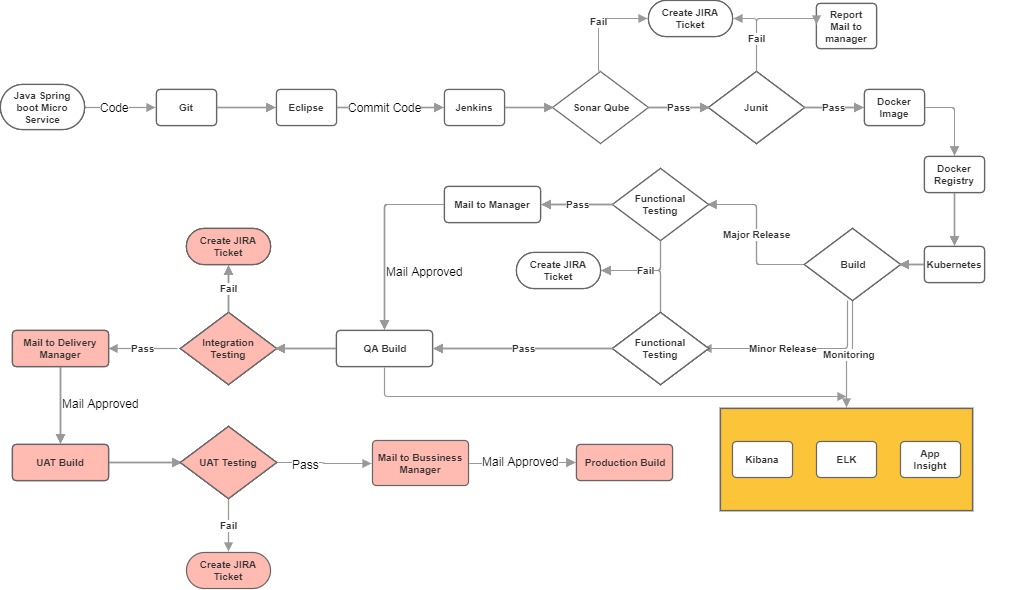
#### Deployment

* Docker image has been built for micro services using Dockerfile.
* The images have been pushed to azure container registry using credentials.
* The images has been pulled from azure container registry and deployed to azure kubernetes cluster.

#### Deployment Strategies

* Blue/Green deployment reduces downtime and risk by running two identical production environments and at any time, only one of the environments is live, with the live environment serving all production traffic.
* Gated Deployment enables secured code check in which doesn’t get pushed to other environments unless it passes a set of test and also ensures Quality delivery in less turnaround time.

#### Architecture Flow:



## Activity 5: API Management

### Scenario 1: Design APIs for Microservices development

* Top down approach is implemented for micro services development. (i.e) Design APIs first and code next.
* Swagger in place to create API specifications.
* Each one of the API pecification like URL, Method, Models, Error are defined in Swagger Editor in YAML or JSON format.
* The designed specifications are viewed and documented using Swagger UI.
* Each method (get, put, post, delete) is expandable in Swagger UI and give full description of the API.
* The methods are tested by passing sample paratmeters and see the response message in Swagger UI.

### Assumptions

1. The overall environment must support 100 Java and 100 .Net developers in both North America and Japan as well as vendor staff developers.